University of Pittsburgh

Visualizations in MATLAB

Week 4

MEMS 1140—Introduction to Programming in Mechanical Engineering

Learning Objectives (L.O.)

At the end of this lecture, you should understand/be able to:

- ❑ The importance of storytelling;
- ❑ How to use compelling visual aids;
- ❑ Write a custom function to visualize the vector cross product.

Table of Contents (ToC)

1 – The Importance of Storytelling

Mechanical engineering problems often have a tangible real-world representation.

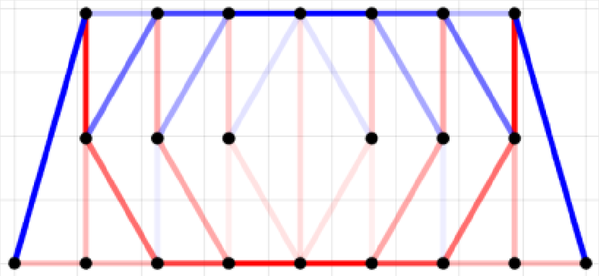
This can make it very easy to find intuitive and useful ways to visualize information.

Code is not enough by itself. We have to learn to tell a story through the code.

2 – Bridge Members Under Load

✓ L.O.1
➩ L.O.2
❑ L.O.3

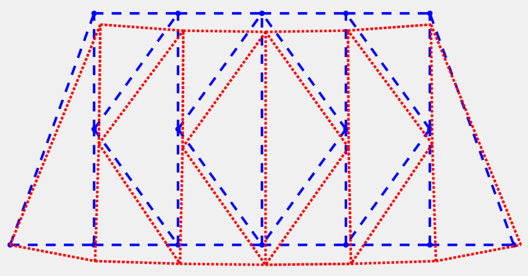
Take this example of a bridge under load.



Each member is highlighted by how much load it carries.

2 – Deformed Bridge Members

Or this example of a bridge under load.



Now, the structure is shown in its deformed state after loading.

2 – Storytelling with Visual Aids

In both of these examples, the subject is the same k-truss under load.

But they both communicate very different information!

Half of our job as engineers is to ***communicate*** our work.

Being able to create compelling visual aids is a powerful skill.

3 – Applying this in Practice

In the Lecture 3.1, we wrote a custom **`crossProduct`** function with two **`1x3`** vectors for its input.

Given that the cross product can be difficult to visualize, let's write a named function to plot everything.

We'll build the function in stages, starting with an outline.

3 – Constructing the Function

The ***input***: two `1x3` vectors, just like the `crossProduct` function from Lecture 3.1.

The ***purpose***: visualize the two input vectors and the one output vector from our custom `crossProduct` function.

And the ***output***: a 3D graph, which shows all three vectors with a labeled legend.

3 – Constructing the Function Outline

Let's first create an outline to follow when building the function:

```
function [] = visualizeCrossProduct(v1, v2)
    .
    .
    .
    .
    .
end
```

Named Function (.m)

3 – Constructing the Function Outline

✓ L.O.1
✓ L.O.2
⇨ L.O.3

Step 1: evaluate the cross product between the input vectors

```
function [] = visualizeCrossProduct(v1, v2)
  % [ ] 1. Evaluate the cross product v1 x v2
  .
  .
  .
  .
end
```

Named Function (.m)

3 – Constructing the Function Outline

✓ L.O.1
✓ L.O.2
⇨ L.O.3

Step 2: plot the *first* input vector

```
function [] = visualizeCrossProduct(v1, v2)
  % [ ] 1. Evaluate the cross product v1 x v2
  % [ ] 2. Plot vector v1
  .
  .
  .
end
```

Named Function (.m)

3 – Constructing the Function Outline

✓ L.O.1
✓ L.O.2
➪ L.O.3

Step 3: plot the ***second*** input vector

```
function [] = visualizeCrossProduct(v1, v2)
  % [ ] 1. Evaluate the cross product v1 x v2
  % [ ] 2. Plot vector v1
  % [ ] 3. Plot vector v2
  .
  .
end
```

Named Function (.m)

3 – Constructing the Function Outline

✓ L.O.1
✓ L.O.2
⇨ L.O.3

Step 4: plot the output vector

```
function [] = visualizeCrossProduct(v1, v2)
  % [ ] 1. Evaluate the cross product v1 x v2
  % [ ] 2. Plot vector v1
  % [ ] 3. Plot vector v2
  % [ ] 4. Plot vector (v1 x v2)
  .
end
```

Named Function (.m)

3 – Constructing the Function Outline

Step 5: format the graph

```
function [] = visualizeCrossProduct(v1, v2)
  % [ ] 1. Evaluate the cross product v1 x v2
  % [ ] 2. Plot vector v1
  % [ ] 3. Plot vector v2
  % [ ] 4. Plot vector (v1 x v2)
  % [ ] 5. Format the graph
end
```

Named Function (.m)

3 – Evaluating the Cross Product

Step 1 just uses the custom function written in Lecture 3.1:

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  % [ ] 2. Plot vector v1
  % [ ] 3. Plot vector v2
  % [ ] 4. Plot vector (v1 x v2)
  % [ ] 5. Format the graph
end
```

Named Function (.m)

3 – The `quiver3` Command

For plotting each vector (Steps 2-4), we will use **`quiver3`**.

The 6 inputs to **`quiver3`** are as follows:

```
quiver3(x_start,          ,          ,          ,          )
```

3 – The quiver3 Command

✓ L.O.1
✓ L.O.2
⇨ L.O.3

For plotting each vector (Steps 2-4), we will use **quiver3**.

The 6 inputs to **quiver3** are as follows:

```
quiver3(x_start, y_start,          ,          ,          ,          )
```

3 – The `quiver3` Command

✓ L.O.1
✓ L.O.2
➪ L.O.3

For plotting each vector (Steps 2-4), we will use **`quiver3`**.

The 6 inputs to **`quiver3`** are as follows:

```
quiver3(x_start, y_start, z_start,          ,          ,          )
```

3 – The `quiver3` Command

For plotting each vector (Steps 2-4), we will use `quiver3`.

The 6 inputs to `quiver3` are as follows:

```
quiver3(x_start, y_start, z_start, x_length,           ,           )
```

3 – The quiver3 Command

For plotting each vector (Steps 2-4), we will use **quiver3**.

The 6 inputs to **quiver3** are as follows:

```
quiver3(x_start, y_start, z_start, x_length, y_length,          )
```

3 – The `quiver3` Command

For plotting each vector (Steps 2-4), we will use **`quiver3`**.

The 6 inputs to **`quiver3`** are as follows:

```
quiver3(x_start, y_start, z_start, x_length, y_length, z_length)
```

We will consider the **`start`** point to be **`(0,0,0)`** for simplicity.

Then, plotting the first vector **`v1`** is as follows:

```
quiver3(0, 0, 0, v1(1), v1(2), v1(3))
```

3 – Plotting Each Vector

✓ L.O.1
✓ L.O.2
⇨ L.O.3

Plotting all three vectors in Steps 2-4 is as follows:

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure    % create a figure object
  .
  .
  .
  .
  % [ ] 5. Format the graph
end
```

Named Function (.m)

3 – Plotting Each Vector

✓ L.O.1
✓ L.O.2
⇨ L.O.3

Plotting all three vectors in Steps 2-4 is as follows:

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on   % prevent each 'quiver3' from overwriting the figure.
  .
  .
  .
  % [ ] 5. Format the graph
end
```

Named Function (.m)

3 – Plotting Each Vector

Plotting all three vectors in Steps 2-4 is as follows:

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  quiver3(0, 0, 0, v1(1), v1(2), v1(3));    % plot v1
  .
  .
  % [ ] 5. Format the graph
end
```

Named Function (.m)

3 – Plotting Each Vector

Plotting all three vectors in Steps 2-4 is as follows:

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  quiver3(0, 0, 0, v1(1), v1(2), v1(3));
  quiver3(0, 0, 0, v2(1), v2(2), v2(3));     % plot v2
  .
  % [ ] 5. Format the graph
end
```

Named Function (.m)

3 – Plotting Each Vector

Plotting all three vectors in Steps 2-4 is as follows:

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  quiver3(0, 0, 0, v1(1), v1(2), v1(3));
  quiver3(0, 0, 0, v2(1), v2(2), v2(3));
  quiver3(0, 0, 0, c(1), c(2), c(3));        % plot v1 x v2
  % [ ] 5. Format the graph
end
```

Named Function (.m)

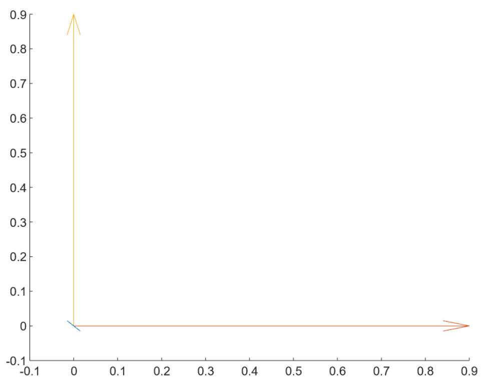
3 – Plotting Each Vector


```
>> v1 = [1,0,0];
>> v2 = [0,1,0];
>> visualizeCrossProduct(v1, v2)
```

Command Window

This graph is not sufficient.

It needs to be formatted.



3 – Changes to Make

In order to make the figure more presentable, we will make the following adjustments:

- Color each vector;
- Increase line width;
- View in 3D;
- Implement a uniform axis aspect ratio;
- Change the axis markers;
- Increase font size;
- Add a Legend;
- Add box markers;
- Add axis labels.

3 – Coloring Each Vector

To color each vector, a color code can be added to the **`quiver3`** command:

```
quiver3(0, 0, 0, v1(1), v1(2), v1(3), 'b'); % color v1 blue
.
.
```

3 – Coloring Each Vector

To color each vector, a color code can be added to the **quiver3** command:

```
quiver3(0, 0, 0, v1(1), v1(2), v1(3), 'b');
quiver3(0, 0, 0, v2(1), v2(2), v2(3), 'g'); % color v2 green
.
```

3 – Coloring Each Vector

To color each vector, a color code can be added to the **quiver3** command:

```
quiver3(0, 0, 0, v1(1), v1(2), v1(3), 'b');
quiver3(0, 0, 0, v2(1), v2(2), v2(3), 'g');
quiver3(0, 0, 0, c(1), c(2), c(3), 'r');    % color c red
```

3 – Increasing Line Width

✓ L.O.1
✓ L.O.2
⇨ L.O.3

The line specification **'LineWidth'** can be used to increase the thickness of each vector within the **quiver3** command:

```
quiver3(0, 0, 0, v1(1), v1(2), v1(3), 'b', 'LineWidth', 2);
quiver3(0, 0, 0, v2(1), v2(2), v2(3), 'g', 'LineWidth', 2);
quiver3(0, 0, 0, c(1), c(2), c(3), 'r', 'LineWidth', 2);
```

As an aside, name-value pairs, like (**'LineWidth'**, **2**) above, are very common in MATLAB.

3 – Viewing in 3D

✓ L.O.1
✓ L.O.2
➩ L.O.3

Adding the **`view(3)`** command displays the graph in 3D:

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...    % quiver3 commands hidden for space
  .
end
```

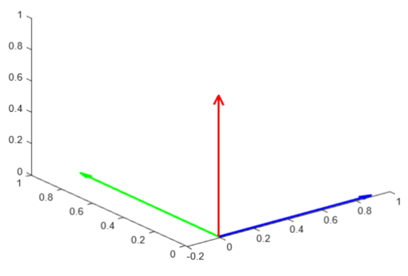
Named Function (.m)

3 – Viewing in 3D

Adding the **`view(3)`** command displays the graph in 3D:

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...    % quiver3 commands hidden for space
  view(3)
end
```

Named Function (.m)



3 – Adjusting Aspect Ratio

`daspect([x,y,z])` specifies an aspect ratio for the axes.

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...    % quiver3 commands hidden for space
  view(3)
  .
end
```

Named Function (.m)

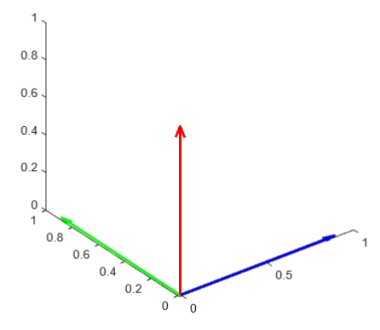
3 – Adjusting Aspect Ratio

✓ L.O.1
✓ L.O.2
➩ L.O.3

`daspect([x,y,z])` specifies an aspect ratio for the axes.

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...    % quiver3 commands hidden for space
  view(3)
  daspect([1,1,1])
end
```

Named Function (.m)



3 – Changing Axis Ticks

✓ L.O.1
✓ L.O.2
➪ L.O.3

xticks, **yticks**, and **zticks** place markers on the axes:

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...   % quiver3 commands hidden for space
  view(3)
  daspect([1,1,1])
  .              .              .
end
```

Named Function (.m)

3 – Changing Axis Ticks

✓ L.O.1
✓ L.O.2
➪ L.O.3

xticks, **yticks**, and **zticks** place markers on the axes:

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...   % quiver3 commands hidden for space
  view(3)
  daspect([1,1,1])
  xticks([0,1]);  .               .
end
```

Named Function (.m)

3 – Changing Axis Ticks

✓ L.O.1
✓ L.O.2
⇨ L.O.3

`xticks`, `yticks`, and `zticks` place markers on the axes:

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...    % quiver3 commands hidden for space
  view(3)
  daspect([1,1,1])
  xticks([0,1]); yticks([0,1]); .
end
```

Named Function (.m)

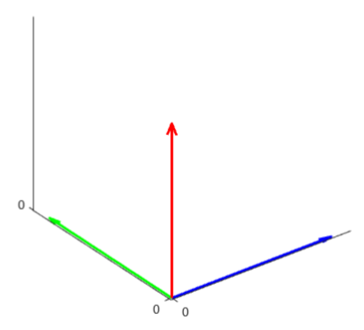
3 – Changing Axis Ticks

✓ L.O.1
✓ L.O.2
➩ L.O.3

`xticks`, `yticks`, and `zticks` place markers on the axes:

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...   % quiver3 commands hidden for space
  view(3)
  daspect([1,1,1])
  xticks([0,1]); yticks([0,1]); zticks([0,1]);
end
```

Named Function (.m)



3 – Changing Font Size

`set(gca, ...)` adjusts properties of the current axes.

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...   % quiver3 commands hidden for space
  view(3)
  daspect([1,1,1])
  xticks([0,1]); yticks([0,1]); zticks([0,1]);
  .
end
```

Named Function (.m)

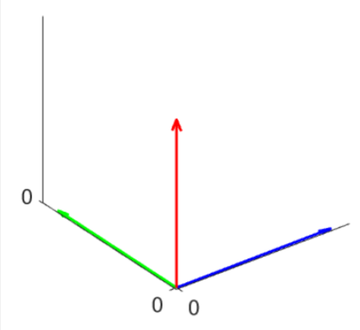
3 – Changing Font Size

✓ L.O.1
✓ L.O.2
➪ L.O.3

`set(gca, ...)` adjusts properties of the current axes.

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...   % quiver3 commands hidden for space
  view(3)
  daspect([1,1,1])
  xticks([0,1]); yticks([0,1]); zticks([0,1]);
  set(gca, 'FontSize', 16);
end
```

Named Function (.m)



3 – Adding a Legend

✓ L.O.1
✓ L.O.2
➩ L.O.3

```
legend('label 1', 'label 2', ...)
```

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...   % quiver3 commands hidden for space
  ...   % prior formatting hidden for space
  .
end
```

Named Function (.m)

3 – Adding a Legend


```
legend('label 1', 'label 2', ...)
```

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...    % quiver3 commands hidden for space
  ...    % prior formatting hidden for space
  legend(    ,      ,          );
end
```

Named Function (.m)

3 – Adding a Legend

✓ L.O.1
✓ L.O.2
➩ L.O.3

`legend('label 1', 'label 2', ...)`

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...    % quiver3 commands hidden for space
  ...    % prior formatting hidden for space
  legend('v1',      ,            );
end
```

Named Function (.m)

3 – Adding a Legend

✓ L.O.1
✓ L.O.2
➪ L.O.3

```
legend('label 1', 'label 2', ...)
```

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...    % quiver3 commands hidden for space
  ...    % prior formatting hidden for space
  legend('v1', 'v2',           );
end
```

Named Function (.m)

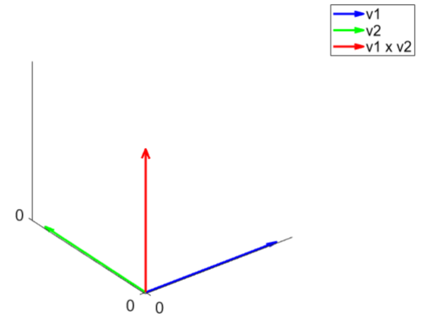
3 – Adding a Legend

✓ L.O.1
✓ L.O.2
➪ L.O.3

```
legend('label 1', 'label 2', ...)
```

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...    % quiver3 commands hidden for space
  ...    % prior formatting hidden for space
  legend('v1', 'v2', 'v1 x v2');
end
```

Named Function (.m)



3 – Adding Box Lines

box and **grid** refer to the plot box and grid lines:

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...    % quiver3 commands hidden for space
  ...    % prior formatting hidden for space
  .
  .
end
```

Named Function (.m)

3 – Adding Box Lines

box and **grid** refer to the plot box and grid lines:

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...    % quiver3 commands hidden for space
  ...    % prior formatting hidden for space
  box on
  .
end
```

Named Function (.m)

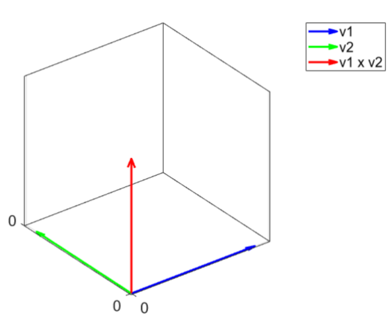
3 – Adding Box Lines

✓ L.O.1
✓ L.O.2
⇨ L.O.3

box and **grid** refer to the plot box and grid lines:

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...    % quiver3 commands hidden for space
  ...    % prior formatting hidden for space
  box on
  grid on
end
```

Named Function (.m)



3 – Adding Axis Labels

✓ L.O.1
✓ L.O.2
➪ L.O.3

`xlabel`, **`ylabel`**, and **`zlabel`** add labels to the axes:

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...    % quiver3 commands hidden for space
  ...    % prior formatting hidden for space
  .
  .
  .
end
```

Named Function (.m)

3 – Adding Axis Labels

xlabel, **ylabel**, and **zlabel** add labels to the axes:

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...    % quiver3 commands hidden for space
  ...    % prior formatting hidden for space
  xlabel('X')
  .
  .
end
```

Named Function (.m)

3 – Adding Axis Labels

✓ L.O.1
✓ L.O.2
➪ L.O.3

`xlabel`, **`ylabel`**, and **`zlabel`** add labels to the axes:

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...    % quiver3 commands hidden for space
  ...    % prior formatting hidden for space
  xlabel('X')
  ylabel('Y')
  .
end
```

Named Function (.m)

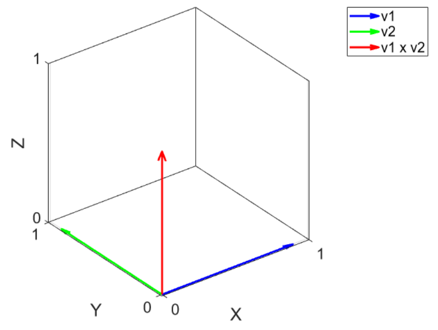
3 – Adding Axis Labels

✓ L.O.1
✓ L.O.2
➪ L.O.3

xlabel, **ylabel**, and **zlabel** add labels to the axes:

```
function [] = visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  ...    % quiver3 commands hidden for space
  ...    % prior formatting hidden for space
  xlabel('X')
  ylabel('Y')
  zlabel('Z')
end
```

Named Function (.m)



3 – Whole Function

✓ L.O.1
✓ L.O.2
⇨ L.O.3

```
function  visualizeCrossProduct(v1, v2)
  c = crossProduct(v1, v2);
  figure
  hold on
  box on; grid on;
  set(gca, 'FontSize', 16);
  quiver3(0, 0, 0, ...
    v1(1), v1(2), v1(3), ...
    'b', 'LineWidth', 2);
  quiver3(0, 0, 0, ...
    v2(1), v2(2), v2(3), ...
    'g', 'LineWidth', 2);
```

Named Function (.m)

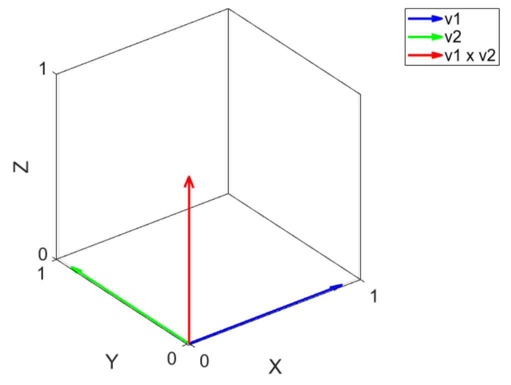
```
  quiver3(0, 0, 0, ...
    c(1), c(2), c(3), ...
    'r', 'LineWidth', 2);
  view(3); daspect([1,1,1]);
  xticks([0,1]);
  yticks([0,1]);
  zticks([0,1]);
  legend('v1', 'v2', 'v1 x v2');
  xlabel('X');
  ylabel('Y');
  zlabel('Z');
end
```

Named Function (.m)

3 – Visualize $\hat{\imath} \times \hat{\jmath}$


```
>> v1 = [1,0,0];
>> v2 = [0,1,0];
>> visualizeCrossProduct(v1, v2)
```

Command Window

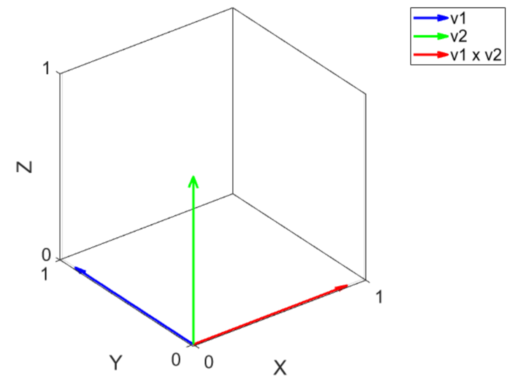


3 – Visualize $\hat{\jmath} \times \hat{k}$

✓ L.O.1
✓ L.O.2
➪ L.O.3

```
>> v1 = [0,1,0];
>> v2 = [0,0,1];
>> visualizeCrossProduct(v1, v2)
```

Command Window

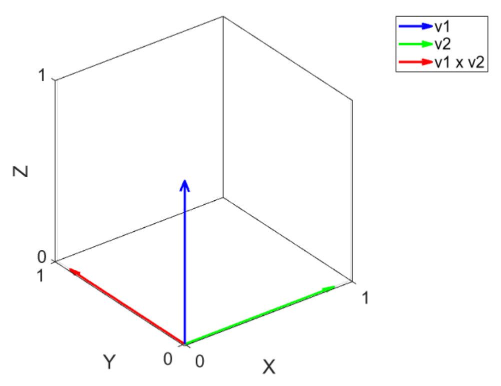


3 – Visualize $\hat{k} \times \hat{\imath}$

✓ L.O.1
✓ L.O.2
➩ L.O.3

```
>> v1 = [0,0,1];
>> v2 = [1,0,0];
>> visualizeCrossProduct(v1, v2)
```

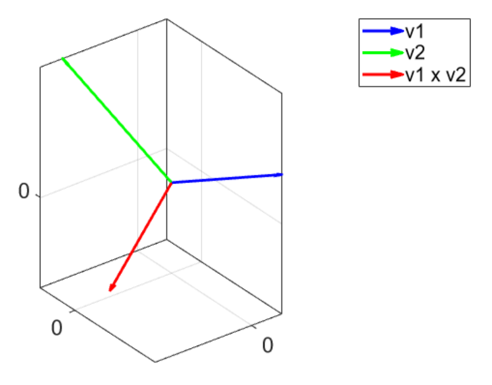
Command Window



3 – Visualize Random Vectors


```
>> v1 = [0.2,-0.7,0.3];
>> v2 = [-0.5,0.3,0.8];
>> visualizeCrossProduct(v1, v2)
```

Command Window



4 – Summary

✓ L.O.1
✓ L.O.2
✓ L.O.3

This lecture covered:

✓ The importance of storytelling

As mechanical engineers, we solve problems *for a reason*. It is necessary to effectively communicate what we have done.

✓ Compelling visual aids

If done well, visual outputs from programs can be extremely valuable in conveying the program's results.

4 – Summary

✓ L.O.1
✓ L.O.2
✓ L.O.3

✓ Writing a custom visualization function for the vector cross product

By using the **`quiver3`** command and formatting specifications like **`LineWidth`**, **`xticks`**, **`legend`**, and more, we plotted all three vectors in the cross product in a visually clear way.